RL-TR-96-255
Final Technical Report
March 1997

# SUPPORTING A SECURE DBMS ON THE DTOS MICROKERNEL (SDDM)

Secure Computing Corporation

Spence Minear, Dick O'Brien, and Lynn Te Winkel

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**Rome Laboratory**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-255 has been reviewed and is approved for publication.

APPROVED: Mary L. Denz

MARY L. DENZ
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO, Chief Scientist
Command, Control, & Communications Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

**1. AGENCY USE ONLY** *(Leave blank)*

**2. REPORT DATE**
March 1997

**3. REPORT TYPE AND DATES COVERED**
Final Jul 95 - Aug 96

**4. TITLE AND SUBTITLE**
SUPPORTING A SECURE DBMS ON THE DTOS MICROKERNEL (SDDM)

**5. FUNDING NUMBERS**
C - F30602-95-C-0244
PE - 33401G
PR - 1068
TA - 01
WU - P1

**6. AUTHOR(S)**
Spence Minear
Dick O'Brien
Lynn Te Winkel

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Secure Computing Corporation
2675 Long Lake Road
Roseville, MN 55113

**8. PERFORMING ORGANIZATION REPORT NUMBER**
N/A

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Rome Laboratory/C3AB
525 Brooks Road
Rome, NY 13441-4505

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
RL-TR-96-255

**11. SUPPLEMENTARY NOTES**
RL Project Engineer: Mary L. Denz/C3AB/315-330-3241

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

The problem addressed in this final technical report deals with hosting a secure database management system (DBMS) on a security-enhanced microkernel architecture. This problem is considered from both an operational issues standpoint and a security issues standpoint.

The operational issues address how well microkernel features meet the operational needs of a DBMS. Operational issues discussed include threaded execution, schedulers, buffer managers and file systems. The three microkernels considered in this report are Mach/DTOS, Flux/Fluke, and LOCK and LOCK6.

The security issues address how well microkernel security features can assist in implementing a secure DBMS on a security-enhanced microkernel. Of particular interest is how the DTOS access control mechanisms can be used to provide higher assurance for DBMS access controls. Both relational and object-oriented DBMSs are discussed.

**14. SUBJECT TERMS**
Multilevel Secure Database Management System, Microkernel, Mach, DTOS, Flux/Fluke, Lock

**15. NUMBER OF PAGES**
52

**16. PRICE CODE**

**17. SECURITY CLASSIFICATION OF REPORT**
UNCLASSIFIED

**18. SECURITY CLASSIFICATION OF THIS PAGE**
UNCLASSIFIED

**19. SECURITY CLASSIFICATION OF ABSTRACT**
UNCLASSIFIED

**20. LIMITATION OF ABSTRACT**
SAR

*Abstract*

This report documents the results of the Supporting a Secure DBMS on the DTOS Microkernel (SDDM) contract, Contract Number F30602-95-C-0244, funded by Rome Laboratory. The objective of the SDDM program was to investigate issues involved with supporting a distributed database management system (DBMS) on Secure Computing Corporation's (SCC's) Distributed Trusted Operating System (DTOS) microkernel using the DTOS control mechanisms.

Key words: DBMS, security, microkernel, operating system

# Contents

# List of Figures

*Section* 1

# Scope

## 1.1 Identification

This report documents the results of the Supporting a Secure DBMS on the DTOS Microkernel (SDDM) contract, Contract Number F30602-95-C-0244, funded by Rome Laboratory.

## 1.2 Program Overview

### 1.2.1 Objectives Summary

The objective of the SDDM program was to investigate issues involved with supporting a distributed database management system (DBMS) on Secure Computing Corporation's (SCC's) Distributed Trusted Operating System (DTOS) microkernel using the DTOS control mechanisms. The work made use of the preliminary design project done by MITRE Corporation, the results of which are documented in MITRE's report, MLS Microkernel DBMS Design Analysis [18].

### 1.2.2 Results Summary

The issues involved with supporting a DBMS on the DTOS microkernel were partitioned into two areas: operational and security. The operational issues dealt mainly with standard operating system support, such as scheduling, buffer management and file systems. DTOS, LOCK6 and Flux/Fluke were evaluated in four areas identified in the MITRE report relative to their ability to provide the type of support in these areas that a DBMS needs. This evaluation is contained in Section 3. The security issues dealt with using the standard DTOS control mechanisms to help provide DBMS access control and considered how enforcement of the DBMS security policy might be partitioned between DBMS controls and DTOS controls. The results of this investigation are contained in Section 4.

## 1.3 Document Overview

The document structure is as follows:

- Section 1, **Scope**, defines the scope of the document and provides a program overview and this overview description of the document.
- Section 2, **General Problem Description**, first discusses the SDDM program goals and the general problem of hosting a DBMS on a microkernel architecture. It then gives a technology summary which includes discussions on microkernels, including Mach, DTOS, Flux/Fluke, and LOCK and LOCK6, and database systems, including relational and object-oriented DBMSs.

- Section 3, **Microkernel Operational Issues and DBMSs**, describes operational issues having to do with hosting a DBMS on a microkernel architecture. It includes discussions of Mach/DTOS, Flux/Fluke, and LOCK6 operational issues.
- Section 4, **Microkernel Security Issues and DBMSs**, describes security issues having to do with hosting a DBMS on a microkernel architecture. It includes a discussion of DTOS microkernel security controls, and then discusses how microkernel security controls can be applied to DBMSs.
- Section 5, **Lessons Learned and Future Directions**, contains a summary of lessons learned on the SDDM program, and discusses future directions of the technology discussed in this report and possibilities for future work in the area of supporting a secure DBMS on a microkernel system.
- Section 6, **Notes**, contains an acronym list.
- Appendix A, **Bibliography**, contains a bibliography of referenced documents.

*Section* 2

# General Problem Description

This section discusses the general problem that the SDDM program focused on. It also gives a technology summary of microkernels and DBMSs, providing information that is useful for understanding the remainder of the report.

## 2.1 Discussion of Program Goals

The goal of the SDDM program was to investigate issues involved with supporting a distributed database management system (DBMS) on Secure Computing Corporation's (SCC's) Distributed Trusted Operating System (DTOS) microkernel using DTOS control mechanisms. The SDDM work made use of a companion DBMS study and preliminary design project done by MITRE Corporation.

The original objectives included:

- Investigating a minimal set of operating system modules that would need to be implemented as servers on the DTOS microkernel to support MITRE's DBMS design.
- Investigating how these servers would work together to enforce the overall security policy of the DBMS.
- Describing how the DTOS security mechanisms can be used to enforce the individual server policies.

After discussions with MITRE on their work, the focus of the first objective changed slightly to discuss the ability of DTOS to support particular features that would benefit implementing a DBMS. Rather than consider particular servers, this report then discusses general issues involved with using the DTOS security mechanisms to support a DBMS application.

## 2.2 Problem Statement

The problem addressed in this report deals with hosting a secure DBMS system on a security-enhanced microkernel architecture. This problem is considered from both an operational issues standpoint (see Section 3) and a security issues standpoint (see Section 4).

The operational issues section addresses how well microkernel features meet the operational needs of a DBMS (as identified by Stonebraker [25] and MITRE [18]). Operational issues discussed include threaded execution, schedulers, buffer managers, and file systems. The three microkernels considered in this report are Mach/DTOS (see Section 2.3.1.1 and Section 2.3.1.2), Flux/Fluke (see Section 2.3.1.3), and LOCK and LOCK6 (see Section 2.3.1.4).

The security issues section addresses how well microkernel security features can assist in implementing a secure DBMS on a security-enhanced microkernel. Of particular interest is how the DTOS access control mechanisms can be used to provide higher assurance for DBMS access controls. Both relational and object-oriented DBMSs are discussed.

## 2.3 Technology Summary

This section will describe microkernel and database systems, focusing on the material that is necessary for an understanding of the remainder of this report.

### 2.3.1 Microkernels

In general an "Operating System" is the collection of software programs that make computer hardware useful. To a typical computer user it provides an environment in which the user can execute programs and manage data. For computer systems that provide services to more than one user, either simultaneously or in sequence, the role of the operating system expands to not only provide operational support but also to provide some level of support for the integrity of itself and the security of the user's data.

The primary design concept for support of both integrity and security is the idea of an Operating System kernel that operates in a processor privileged state. All modern processors provide at least two modes of operation. They are frequently referred to as either, "Privileged Mode and User Mode" or "Supervisor Mode and User Mode". The difference between these two modes varies with each processor, but in general it is that there are a small set of privileged instructions that can be executed only when the processor is operating in privileged mode. Examples of these instructions include changing the state of the Memory Management Unit (MMU), thereby controlling the memory space in which a program is operating, changing the state of the processor itself by modifying the various processor status and control registers, and managing the hardware through control of processor interrupts.

Typically all of the user programs operate in user mode, while a majority of the operating processing is done while operating in privileged mode. The body of operating system software that operates in privileged mode is referred to as the "operating system kernel" or just the "kernel". For most modern multiuser operating systems, the kernel includes a wide range of operating system functions including: all process management functions, file system functions, network protocol processing, and hardware management. This tends to be a large body of software that is not always well structured. Thus such an OS kernel is frequently referred to as a "monolithic" OS kernel. This design approach has lead to both operational and security problems. Operational problems occur because there is no way to confine the impact of errors. Since all OS functions operate in the same environment and have full access to all system data and system facilities, an error in one part of the system can result in the all too frequent "system crash".

For the security community such large, monolithic OS kernels present a problem because they violate many of the basic tenets of good security systems, the most notable of which is least privilege. Least privilege calls for a system design in which each critical function executes in an environment in which it has access to only the data and facilities that it requires. The monolithic OS approach provides no support for separating the processing elements either for operational integrity, operational correctness, or system security reasons.

In the late 1980's, some operating system researchers responded to these problems by developing the concept of a microkernel based Operating System. The core concept is to identify the minimum set of operating system functions that need to operate in processor privileged mode and implement them as a smaller system kernel, thus the name microkernel. The remainder of the normal operating system functions are then provided in operating system processes which operate in the processor's user mode and utilize the services of the microkernel much as traditional applications utilize the monolithic operating system services. This approach has several benefits.

- The amount and complexity of code operating in the processor's privilege mode is greatly decreased.
- The interaction between other operating system functions is controlled much better, which greatly decreases the potential for unexpected interactions between different OS functions.
- The system becomes more flexible and easier to maintain and expand because new features can be added without the the need to change the kernel.

The one overriding technical issue for microkernel based systems has been performance. When functions are separated out into separate user mode programs the ability to communicate and exchange information efficiently is significantly impacted. When loosely coupled functions co-reside in a monolithic kernel, they can read shared data structures and call each other's services with simple efficient subroutines. The same is not true in a microkernel based system. In such a system, it is necessary to use a higher cost Inter-Process Communication (IPC) mechanism or Remote Procedure Call (RPC) mechanism. Thus, though taking existing tightly coupled operating system functions and breaking them up into separate user mode processes has obvious security and integrity benefits, it can present a significant performance problem.

The remainder of this section provides an introduction to several microkernel based systems that have been assessed both directly and indirectly as part of this effort.

#### 2.3.1.1 Mach

This section will give an overview of the Mach microkernel.

Information in this section is taken from the paper, Providing Policy Control Over Object Operations in a Mach Based System [17] by Spencer E. Minear.

Mach is a microkernel providing a set of basic facilities for use by operating systems and other applications. It is designed around an Inter-Process Communication (IPC) facility based on a port. Mach and systems built on Mach utilize Object Oriented Design concepts by building on Mach's port abstraction. A port can be used as an object handle, through which object methods are invoked or object service requests are made. In addition to ports, Mach provides several other types of kernel objects including tasks, threads, and memory cache objects [14]. Tasks provide an environment in which all processing is done. All processing is done by a specific thread and each thread is bound to a single task. Memory cache objects provide memory in which to store and manipulate data. In addition to these basic objects, the Mach kernel supports other objects such as devices, processors, and the kernel itself. Each of the non-port objects has one or more associated ports that are used to represent the object and through which all operations on the object are initiated.

An examination of the basic Mach structure shows that it is made up, primarily, of two parts; the IPC services provider and the set of object servers implementing the services of the non-port kernel objects. Mach uses its own IPC facilities to provide tasks access to its other types of objects. To request an operation on a non-port kernel object, a task sends a request to that object via its associated port. The IPC send operation is processed by the kernel the same as any other send operation. When the send processing recognizes that the target object is a kernel object, control is transferred to that object's kernel server for processing. If the target object is managed by a task external to the kernel, the request is provided to that task via its use of the IPC receive operation on the same port. Figure 1 shows the relationship between the kernel's IPC services, the kernel's object servers, and the object servers that operate as tasks external to the kernel. Communication between clients and servers is provided by communication connections implemented in the kernel IPC services.

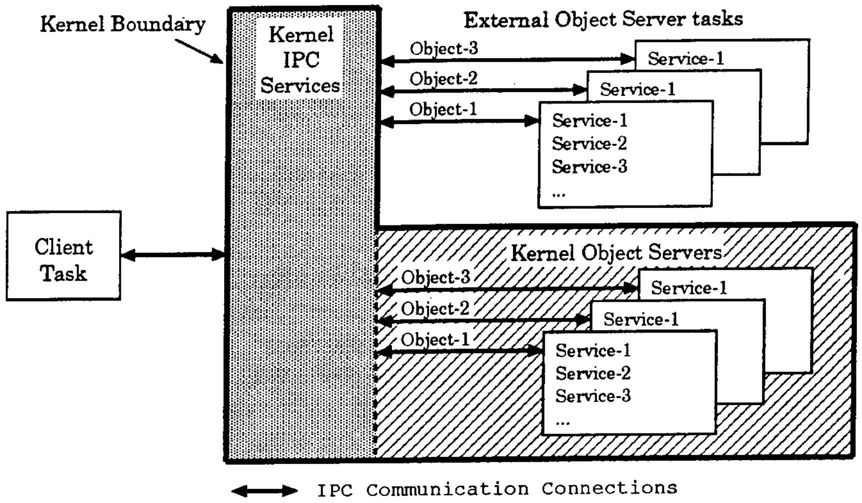

Figure 1: Mach Kernel Structure

2.3.1.2 DTOS This section gives a brief overview of the DTOS program and the DTOS system which is built on the Mach operating system. The majority of this section was taken from the paper, Developing a "Policy Neutral" Control Policy for a Microkernel [6], by Todd Fine and Edward A. Schneider.

The DTOS program is a follow-on to the Distributed Trusted Mach (DTMach) program [5]. The goal of the DTMach program was to develop a design for a distributed, trusted operating system based on the Mach 3.0 microkernel [15, 26]. The DTOS program is exploring the feasibility of the DTMach design through prototyping and study efforts.

One of the goals of the DTOS program is to investigate an approach for developing a microkernel that supports a wide range of security policies. Rather than simply following the guidelines in the Trusted Computer Security Evaluation Criteria (TCSEC) [19] and implementing Discretionary Access Control (DAC) and Multilevel Security (MLS), the DTOS microkernel provides a framework that encompasses these policies as well as others.

The DTOS security architecture supports policy flexibility by separating the making of policy decisions from the enforcement of those decisions. The policy decisions are made by *security servers*. A security server is simply a process executing in the system that makes decisions based on a set of security rules. The enforcement of these decisions is performed by the system component managing the protected objects.

Since the DTOS security architecture is significantly different than previously implemented security architectures, approaches used to develop security policies in the past were inadequate for DTOS. In particular, an approach was needed that allowed the requirements on how the microkernel enforced access to be separated from the requirements on how a given security server made access decisions. The developed approach relies on composablilty theory [1] to allow each of these sets of requirements to be specified separately and then integrated into an overall system policy.

2.3.1.3 Flux/Fluke This section gives a brief overview of the Flux/Fluke system being developed at the University of Utah. Flux is the name of the overall system, while Fluke is the name of the microkernel. This information in this section is based on the paper, Microkernels Meet Recursive Virtual Machines [8], by Brian Ford, Mike Hibler, Jay Lepreau, Patrick Tullmann, Godmar Back, Shantanu Goel, and Steven Clawson.

The Flux/Fluke system consists of the microkernel running on a hardware platform together with support for a virtual machine architecture. The virtual machine architecture allows the development of virtual machine that can run applications or recursively support other virtual machines. The microkernel API provides simple memory management, scheduling and IPC primitives. The virtual machine architecture uses a set of IPC based "common protocols" for communication between the virtual machines.

The Fluke kernel primitives include:

- threads: for flow of control.
- spaces, regions and mappings: for thread address space management. Spaces define a thread's address space. Regions are used to export memory from a space, and mappings are used to import memory to a space.
- ports, port sets and port references: for IPC. IPC is based on a capability model like Mach's. A port represents the server side of a communication channel and a port reference the client side.
- mutexes and condition variables: for synchronization of access to shared memory.

A minimal form of scheduling is provided by allowing threads to donate their CPU time to other threads, based on some higher level scheduling policy, or to request more time.

The Fluke kernel contains no special security mechanisms; all low-level support for security is integrated into the other primitives exported by the kernel. In particular, there is currently no support for security control based on a subject's security context. Research is currently going on in this area to determine the best way to add such control within the context of the Flux architecture.

2.3.1.4 LOCK and LOCK6 This section will give an overview of LOCK6. The LOCK TCB, developed by Secure Computing Corporation during the Logical Coprocessing Kernel (LOCK) program, is a secure operating system with a microkernel architecture. The fundamental design goal of the LOCK system was to design a highly secure minimal system. The TCB itself had only 23 entry points and provided services for a very limited set of basic TCB entities. These services included subject management such as scheduling and signaling, data storage object management, memory managment, and basic hardware functions such as time and access to device drivers.

LOCK6 is the name of the second generation LOCK TCB being developed by SCC for use in the latest versions of the Secure Network Server (SNS) system. Though the implementation is quite different, there are many aspects of the LOCK6 system that are similar to the original LOCK system. Like the original LOCK system, the code which operates in the processor's supervisor state was designed to be minimal both in size and function. Other OS functions are implemented as user state processes which, as in LOCK, communicate with each other and other processes utilizing the basic communication facilities supported by the supervisor state software.

There are two major differences between LOCK and LOCK6. The first is that the fundamental control point in LOCK6 is the message based Inter-Process Communication (IPC) facility, not memory mapped objects as in LOCK. The second is that the TCB is significantly larger in that it has been expanded to support a POSIX interface rather than the original 23 entry interface provided by LOCK. LOCK6 continues the use of a microkernel architecture and very strong separation of the security policy enforcement from the security policy logic.

2.3.1.4.1 LOCK6 Architecture As with other microkernel systems, LOCK6 was designed so that a minimum of processing is done in the processor's supervisor state. Unlike other microkernel systems, LOCK6 has very strong and flexible control facilities integrated into the TCB. The LOCK6 kernel provides support for the following basic objects and their supporting operations:

- Processes,
- Threads,
- Memory objects, and
- IPC channels

Other higher level operating system objects such as files, sockets, and devices are implemented in various Operating System processes which operate in the processor's user state.

As with Mach and Fluke based systems, nearly all operating system level operations in LOCK6 are accessible only via the IPC facilities. This includes access to files, memory objects, devices, and higher level communications channels such as sockets. This mapping of higher level objects to IPC channels is all done within POSIX or other library interfaces, making the details of the implementation transparent to the applications.

### 2.3.2 Database Systems

This section provides an overview of current commercial database system architectures and security capabilities. Both relational and object oriented systems are discussed.

2.3.2.1 Relational Database Systems Relational database systems (RDBMSs) are based on the relational model developed by Codd [3]. The major commercial databases (Oracle, Sybase, Informix, Ingres, DB2) are all RDBMSs. The relational model is based on standard set theory and relational algebra operations. In the relational model, all data (including metadata that defines the database, such as definitions of views and stored procedures) is stored in tables. Each row of a table has a unique identifier, called the key, and represents a record in the table. Each column of the table has a fixed data type and represents a particular attribute of the table. Operations on RDBMSs are performed using a special, standardized, query language called SQL that is built on relational algebra.

For a relational DBMS, the objects to which access needs to be controlled are DBMS related entities. In the discussion that follows, Trusted Oracle 7 (TO7) [4] is used to illustrate the security functionality of a typical relational DBMS.

2.3.2.1.1 Identity Based Access Control (IBAC) For Trusted Oracle 7, the following DBMS "named objects" are defined:

- Tables
- Views
- Stored procedures
- Sequences
- Packages (a collection of functions, procedures, variables, constants and exceptions)

IBAC is enforced on these named objects by the DBMS. Either the owner of the object, or someone granted a special privilege via the WITH GRANT OPTION or WITH ADMIN OPTION, can give an object privilege for a named object to another user or user-group(role). The object privileges supported are:

- ALTER (a table or sequence)
- DELETE (a table or view)
- EXECUTE (a procedure)
- INDEX (a table)
- INSERT (into a table or simple view)
- REFERENCES (modify a table's foreign key integrity constraints)
- SELECT (read a table, view or sequence)
- UPDATE (write a table or simple view)

Roles are defined as sets of users and object privileges can be granted to roles or to individual users.

There is also a set of system privileges that control what special commands a user (or role) can execute. These are usually only granted to system administrators. In TO7 there are over 60 such privileges. They include things like:

- ALTER DATABASE
- CREATE ANY INDEX
- CREATE ROLE
- DROP ANY TABLE
- ALTER ANY TABLE
- GRANT ANY PRIVILEGE.

**2.3.2.1.2 Mandatory Access Control (MAC)** In most commercial secure RDBMSs, MAC is enforced at the granularity of rows. That is, each row in a table has an associated level. This is the standard approach used by Oracle, Sybase and Informix for their trusted DBMSs. The DBMS runs as a trusted subject(s) that enforces the labeling constraints. The trusted subject is responsible for labeling the rows correctly and only retrieving information that the user is cleared to see. The information is stored in files at DBMS High level and every response to a query involves a downgrade by the trusted DBMS server subject. Note that all named objects in the DBMS are contained in tables of some form or other and thus have associated levels. Tables may be multilevel objects. For example, there is a system table that contains all views defined for the database. Each view definition is a row in this table and the level of the view is defined as the level of the corresponding row in the table.

TO7 actually has two modes:

- DBMS MAC mode, in which the DBMS runs as a trusted subject and enforces MAC as described above
- OS MAC mode, in which the database is partitioned by level and the operating system protects information stored in files at different levels.

OS MAC mode is a tcb subset architecture, as described in the TDI [20]. The advantage of OS MAC mode is that on a high assurance operating system, the operating system is responsible for enforcing MAC on the DBMS objects. This provides a higher assurance MAC without having to modify, or add additional assurance to, the DBMS. The disadvantage of OS MAC mode is that it requires the DBMS processes to be multiply instantiated; once at each level. If there are several levels, this can affect performance. Since higher level instances of the DBMS cannot talk to lower level instances, this also makes it more difficult to enforce integrity constraints across the entire multilevel database. Also, even in OS MAC mode, the DBMS is responsible for enforcing IBAC on the DBMS named objects.

**2.3.2.1.3 Client/Server Architectures** Most current commercial RDBMSs are networked based and implement a client/server architecture. The client runs on the user's workstation and communicates with a server program running on the database server machine. The server process can either be dedicated to the particular client, with one server process per client, or can be a multi-threaded server that handles connections from several clients simultaneously. Clients can be required to authenticate themselves to the server, but, in most cases, it is then the server's responsibility to ensure that only data that the client is allowed to see is returned to the client.

**2.3.2.2 Object Oriented Database Systems** Object Oriented Database Systems (OODBMS) are based on the concept of objects rather than relations. Object technology is becoming widely used for software development because it promotes development of modular software that can be easily shared and reused. Object DBMSs are especially useful for supporting complex data types and distributed applications.

While there is no consensus OODBMS model that corresponds to Codd's relational model, there is a published model that several commercial OODBMS vendors have agreed to support [16]. In this model an object is a primitive that has a unique identifier. Objects have associated operations that define their behavior and associated properties (attributes and relationships) that define their state. Objects are organized in a hierarchy of types and subtypes. Subtypes inherit the behavior and state associated with their supertypes.

OODBMSs are closely related to object oriented programming languages such as C++ and SmallTalk. There is no standardized query language that plays the role that SQL does for relational DBMSs. Queries are done either directly via the programming language, or via extensions to the programming language that provide additional query capabilities; in some cases, SQL-based capabilities. The primary features that OODBMSs add to an Object Oriented programming environment include:

- object persistence: so the data is not lost when the program terminates
- shared objects and concurrency control
- transactions
- querying capabilities
- versioning support
- access control and audit
- integrity constraints
- recovery capabilities.

An interesting difference, for security considerations, between an OODBMS and a RDBMS is that most of the OODBMS code executes in the context of the user and being able to share objects involves sharing between users. For most commercial RDBMSs, the server processes execute in the name of a dbms pseudo-user, and, for efficiency, server processes share common data storage areas.

There are currently no commercial MLS OODBMSs. ONTOS is developing a prototype system called the Trusted ONTOS Prototype (TOP) for Rome Laboratory [23]. This is the model we will use in our analysis of OODBMSs and DTOS control mechanisms.

The following features of TOP are of interest to our study:

- Objects are multilevel. Each attribute in an object type is assigned a *visibility* level that identifies the lowest level at which it is visible. Different attributes may have different levels. The level of an attribute in an instantiated object must dominate the *visibility* level.
- A TOP object comprises a set of L-instantiations, each of which consists of the data entered into the object at level L. An object's L-instantiations share a common object id and each is stored at its own level, L.
- A user's access to an object is through a view. Users do not access L-instantiations directly. A user at level L would be presented with the L-view of the object. The L-view of the object is constructed from the L-instantiation and the semantic vector associated with the L-instantiation that defines which attributes should be inherited from a lower level.
- Operations that involve modification to the database at a level lower than the level of the user are handled via a TCB trusted path that queries the user before the operation is performed.

Plans are for TOP to use Discretionary Access Controls to implement separate roles for the DBA and SSO as well as user and group based controls. The details of the TOP DAC mechanism were not available to us at the time of this report.

# *Section* 3 Microkernel Operational Issues and DBMSs

This section discusses several operational issues having to do with hosting a DBMS on a microkernel architecture. These issues were originally raised in a paper by Stonebraker [25] discussing operating system support for DBMSs and were revisited in the MITRE report, MLS Microkernel DBMS Design Analysis [18]. In particular, the MITRE r.port discusses threaded execution, schedulers, buffer managers, and file systems, and describes certain features in these areas that would facilitate supporting a DBMS. The focus of this section is on describing the functionality of three microkernel architectures in these areas and evaluating how well they meet the criteria in the MITRE report. The functionality under evaluation includes both the fundamental design characteristic of the microkernels as well as the ease with which vendor-supplied modules with special functionality can be incorporated into the system. The systems considered are Mach/DTOS, Flux/Fluke, and LOCK6.

## 3.1 Operational Issues Summary

Before looking at the individual systems, the operational issues raised by the MITRE report are summarized in this section. These issues involve four areas: threaded execution, schedulers, buffer managers, and file systems. The particular functionality in each of these areas that would support hosting a DBMS on the system is identified. In certain areas, such as scheduling, buffer managing and file systems, the issue is not whether the microkernel has the desired functionality included in it, but whether it is possible to add modules to the system that would provide the needed functionality.

### 3.1.1 Threaded Execution

As discussed in Section 2.3.2.1, in a client/server architecture there are two common ways in which the binding between client and server can occur. Until recently, the usual method was to have one server for each client and to allow these servers to access a shared data area for caching database information. For systems with a high number of concurrent users, this had the disadvantage of putting a heavy load on the system due to the number of servers and the number of context switches that would have to occur. Multi-threaded servers provide the ability for a single server to handle several clients by using separate threads within the server process for each client.

Most commercial RDBMSs now support multi-threaded servers but to do so efficiently requires support from the operating system. So the ability to support threads within a process is a feature that the microkernel should provide to support DBMSs.

Since the multi-threaded server runs as a single process, all threads within that process have access to the same virtual memory. This implies that the only thing protecting the information that one client thread can access from being accessed by a different client thread is the correctness of the server implementation. An issue that was discussed was whether some separation between threads could be achieved. This might include assigning each thread its own security identifier (SID). A related idea that was discussed was to allow a thread to sub-set the process virtual address space for newly spawned threads in an attempt to provide a degree of virtual memory isolation for threads. The feasibility of these ideas is discussed in Section 3.2.1.

### 3.1.2 Schedulers

Many operating systems provide general purpose scheduling algorithms, such as strict round-robin, that are not efficient for DBMS operation. These algorithms can result in poorer performance due to unnecessary or inappropriate context switches. Microkernel scheduler functionality that would assist in hosting a DBMS on a microkernel include:

- The ability to change the scheduling priority of processes so that DBMS processes could have their priority modified as needed.
- The ability to install and use application specific scheduling algorithms that are tuned for better DBMS performance.

### 3.1.3 Buffer Managers

Buffer managers are used to handle paging of information between secondary storage and main memory. The ability to control aspects of this paging activity would provide additional efficiencies for a DBMS implementation. In particular, the MITRE report identifies the following desired capabilities:

- The ability to install and use page replacement algorithms that are customized for the DBMS application.
- The ability for an application to request that pages be flushed to disk.
- The ability to request selective, temporary, remapping of virtual address spaces.

### 3.1.4 File Systems

In a microkernel based system the file system is normally not part of the microkernel, but rather is integrated as a separate server into the system. This implies that DBMS friendly file servers could be developed that provide specific support for DBMS applications. The type of support that such a file system might provide includes: disk striping, creating per-partition file system servers, and integrating file system access with virtual memory address-space remapping.

## 3.2 Mach/DTOS

This section discusses how Mach and DTOS address the operational issues discussed in Section 3.1.

### 3.2.1 Threaded Execution

Mach/DTOS does support threaded execution. Hence, Mach/DTOS tasks and threads can be used to implement multi-threaded servers. The question is how the Mach/DTOS implementation of threads might assist in the other issues discussed in Section 3.1.1.

In DTOS, security identifiers (SIDs) are associated with tasks, not threads. If a DBMS server is multi-threaded (one thread per user), then microkernel support for separation between the threads would require that each thread have its own security context or SID. So, while DTOS does support multithreaded operation, it does not support the ability to run threads within their own security context. Adding separate security contexts to threads would involve major changes to the Mach/DTOS design and would essentially eliminate the distinction between a thread and a task.

As mentioned in Section 3.1.1, an additional idea discussed was allowing a thread to sub-set the task virtual address space for newly spawned threads as an attempt to provide a degree of virtual memory isolation for threads. In our opinion sub-setting the task virtual address space for newly spawned threads is impractical in Mach/DTOS and would eliminate the fundamental design advantages of Mach threads.

Another DTOS consideration should be included in this section. An important DTOS design criteria is the idea of separating the two areas of policy decision and policy enforcement. The DTOS Security Server is responsible for making policy decisions while the DTOS Trusted Computing Base (TCB) is responsible for enforcing policy decisions. So while the DTOS microkernel cannot feasibly support the association of SIDs with threads, and thus use the DTOS TCB to enforce policy decisions, the DBMS server can. Using the fact that in DTOS the TCB is separate from the Security Server, the DBMS server could bind a SID to each of its threads and then use the Security Server to drive policy decisions. It would then be the DBMS server's responsibility to identify the necessary control points and enforce the policy decisions made by the Security Server.

### 3.2.2 Schedulers

The Mach/DTOS interface calls (taken from the DTOS Kernel Interfaces document [24]) that are relevant to scheduling are the following:

**swtch** Attempts to context switch the current thread off the processor.

**swtch_pri** Attempts to context switch the current thread off the processor. The thread's priority is lowered to the minimum possible value during this time. The priority of the thread will be restored when it is awakened.

**processor_set_max_priority** Sets the maximum scheduling priority for a processor set.

**processor_set_policy_disable** Disables a scheduling policy for a processor set.

**processor_set_policy_enable** Enables a scheduling policy for a processor set.

**task_priority** Sets the scheduling priority for a task.

**thread_switch** Provides low-level access to the scheduler's context switching code.

**thread_max_priority** Sets the maximum scheduling priority for a thread.

**thread_policy** Sets the scheduling policy to apply to a thread.

**thread_priority** Sets the scheduling priority for a thread.

**thread_wire** Marks the thread as "wired". A "wired" thread is always eligible to be scheduled and can consume physical memory even when free memory is scarce.

**thread_depress_abort** Cancels any priority depression effective for a thread caused by a **swtch_pri** or **thread_switch** call.

Using Mach calls such as **task_priority** and **thread_priority**, the scheduling priority of processes can be changed. The concept of an interface for registering alternate scheduling algorithms is harder to meet in Mach/DTOS. This is explained further in the following paragraphs.

The Mach interface implies flexibility and provides a mechanism which allows a total of 32 different policies to be supported, but the implementation is very strongly biased to two simple policies, timeshare and fixed priority. A scheduling policy is bound to a processor set and there is a facility to define which of the "supported scheduling policies" is to be used on a given processor set. Tasks can be assigned to a processor set and thus are impacted by the scheduling policy that is selected for a processor set. The focus of the scheduling algorithm is the thread.

Each processor has 32 run queues, one for each of 32 different priorities. The maximum number is a compile time value. Each processor set has 32 run queues, one for each of 32 different priorities. The maximum number is a compile time value. The normal search pattern is to look on a processor's run queue first and then look on the associated processor's processor set's run queue. When a thread is ready to run it gets put on the run_queue associated by its current computed priority.

In Mach/DTOS there are no facilities to dynamically provide a scheduling algorithm at runtime. Theoretically it is possible to create new scheduling policies and make them effective on a per thread and/or per processor set basis. However, to add a new policy would require changes to the current Mach code.

If the capability to introduce new scheduling algorithms was implemented in DTOS, this could also lead to assurance problems. Introducing unknown code that implements scheduling into the system can introduce new covert channels. Each piece of code that implemented a scheduling algorithm would have to be secured and assured prior to its use in the system.

### 3.2.3 Buffer Managers

Mach/DTOS does not support an interface to request modified page-replacement algorithms. However, using a combination of other Mach/DTOS mechanisms, it may be possible to implement other page-replacement algorithms as needed.

The first Mach/DTOS mechanism that helps in implementing alternate page-replacement algorithms is the **vm_wire** interface call. Wiring is allowed down to a single page or over a defined contiguous region of memory. Wiring faults the requested pages in and fixes them in memory.

The second Mach/DTOS mechanism that helps in implementing alternate page-replacement algorithms is the ability to specify external memory managers. As described in the OSF Mach

Kernel Principles document [15], the Mach kernel allows user mode tasks to provide the semantics associated with the act of referencing portions of a virtual address space. It does this by allowing the specification of an abstract *memory object* that represents the non-resident state of the memory ranges backed by this memory object. The task that implements this memory object (that responds to messages sent to the port that names the memory object) is called a *memory manager.* Mach/DTOS has a default memory manager which is an external memory manager that provides backing storage for anonymous memory. In addition to the default memory manager, it is possible for users to specify external memory managers that are then associated with memory objects. These external memory managers can then be implemented so as to facilitate alternate page-replacement algorithms.

One note on the above paragraph that may dismiss the possibility of implementing alternate page-replacement algorithms by using external memory managers is the following. Under the Physical Memory section of the OSF Mach Kernel Principles document [15], the following statements are made: *The majority of the physical memory of the system forms a single paging pool. This pool of pages forms a cache for the virtual memory system. The set of pages that reside in physical memory at any given time is decided by the page replacement algorithm, implemented in the kernel. Clients have no control over this algorithm (with the exception of the* ***vm_wire*** *call). Even external memory managers have no influence; if they do not respond fast enough to a request to write a page, the default memory manager will be used to move the page from physical memory to system paging storage (except for the clean-in-place mechanism reserved for "trusted" managers).*

Interfaces to allow selective remapping of virtual address spaces can be accomplished in Mach/DTOS by using the out-of-line data option of **mach_msg**. Data can be transferred using this facility on either a copy-on-write basis or a transfer basis. One thing to note is that this is a permanent transfer of data, it is not a **temporary** adjustment of page tables.

### 3.2.4 File Systems

Since the Mach/DTOS microkernel does not include its own file system but provides support for file system servers, it would be possible to implement file systems having the capabilities described in Section 3.1.4.

## 3.3 Flux/Fluke

This section discusses how Flux and Fluke address the operational issues discussed in Section 3.1. This section is necessarily very preliminary since our information on Flux and Fluke is limited to the descriptions in the papers: Microkernels Meet Recursive Virtual Machines [8], Fluke: Flexible $\mu$-kernel Environment Design Principles and Rationale [9], and Fluke: Flexible $\mu$-kernel Environment Application Programming Interface Reference [7].

### 3.3.1 Threaded Execution

Fluke does support threaded execution and as we understand it, threads will be separately identified, in contrast to DTOS where only tasks separately identified. The separate identification of threads addresses some of the issues mentioned in Section 3.1.1. As mentioned Section 3.2.1, an idea was also discussed concerning sub-setting the task virtual address space for newly spawned threads. We believe that Fluke will not allow this, although we do not know this for a fact.

### 3.3.2 Schedulers

The paper, Microkernels Meet Recursive Virtual Machines [8], states that in Fluke, the kernel provides a primitive that supports hierarchical scheduling models. In this model individual threads act as schedulers by donating some of their CPU time to other threads. A scheduling hierarchy is developed in which threads higher in the hierarchy donate time to lower level threads. The individual thread scheduling can be based on a common scheduling algorithm that all threads follow.

The `fluke_thread_state` data structure has a `scheduler_ref` field which is a reference to the port to which the thread sends IPC messages to request CPU time. If NULL, the thread can only run using donated time from another thread.

In the **fluke_thread_set_state** call (set the state of a thread object), there is a parameter called `scheduler_ref`. The reference must be null or must point to a port object. If `scheduler_ref` is non-null, it indicates the reference object to be inserted as the thread's scheduler port reference. If `scheduler_ref` is a null pointer, then the thread's scheduler reference is unchanged.

There is also a `fluke_thread_schedule` call which schedules another thread to run.

It appears that the Fluke scheduling mechanism is fairly flexible and may be able to address some of the issues detailed in Section 3.1.2. From looking at the current Fluke API document [7], it does not appear that the capability exists in Fluke to alter task and thread priorities.

### 3.3.3 Buffer Managers

It appears that there is some flexibility in specifying page replacement policies in Flux/Fluke. Each region has an associated keeper port, which identifies the page fault handler for the region. This would help address the issue concerning the interface to request modified page-replacement algorithms mentioned in Section 3.1.3.

In addition, it appears that interfaces to allow selective remapping of virtual address spaces can be accomplished in Fluke by using mapping objects.

### 3.3.4 File Systems

We do not have enough information about Fluke to comment on their file system mechanisms.

## 3.4 LOCK6

This section discusses how LOCK6 addresses the operational issues discussed in Section 3.1.

### 3.4.1 Threaded Execution

LOCK6 does support the use of threads both within the LOCK6 TCB and in applications. LOCK6 threads make it easier to port existing threaded applications and can help to remove the need for complex software from the applications which keep track of multiple simultaneous activities. However, threads always execute within the context of a single process and inherit their security attributes from the containing process. Thus all threads within a given process have equal access to all files, communication channels and memory accessible to their containing process. In particular, a multi-threaded DBMS server that used separate threads for separate clients would need to enforce its own control over what each thread could access.

### 3.4.2 Schedulers

The LOCK6 system does not make the system scheduler visible to applications, beyond supporting the related POSIX requirements. This approach was chosen because of the following security concerns.

- Scheduling issues can be a significant source of covert channels within a secure system. The LOCK6 philosophy is that anything that can introduce covert channels must be reviewed very carefully, and such functionality is typically left out of the system unless there is a very strong overriding requirement to include it.
- To support efficient scheduling normally requires extra access to and/or modification of the lowest levels of the kernel's operation. The lowest level of the kernel is typically the most security critical part of the system. Any increase in complexity or changes to this area can have significant impact on the overall assurance state of the system.

Thus the LOCK6 system provides very limited access to the scheduling processing.

### 3.4.3 Buffer Managers

The LOCK6 system does not provide any specific support for buffer management beyond the normal access to memory objects as defined by the POSIX standard or a BSD like Unix OS.

### 3.4.4 File Systems

LOCK6 provides a standard POSIX file system interface which has been extended to provide both MLS and Type Enforcement based control over access to files. The LOCK6 architecture has completely separated the file system functions from the system kernel. Thus it is structured to allow the addition of specialized file systems to the running system. However no services have been implemented within the existing file system to allow the mounting of new file systems which may be tuned to support a specific application. Again the major issue is concern for security and/or the usefulness within a highly secure system for a less secure file system.

Section 4

# Microkernel Security Issues and DBMSs

This section describes security issues having to do with hosting a DBMS on a microkernel architecture. It begins by describing the security controls included in Mach, DTOS, and LOCK6. Due to the fact that Flux/Fluke is currently under development and we do not know the security controls that will be added to it, Flux/Fluke is not discussed in this section. This section ends by discussing how the DTOS microkernel security controls could be applied to a DBMS. In particular, the following questions are considered:

- What portions of a DBMS security policy can the DTOS microkernel security controls support?
- What portions of a DBMS security policy must the DBMS support?
- What support can the DTOS microkernel lend to the DBMS to help it support its portion of the security policy?
- How might a DBMS be structured to better take advantage of the DTOS security controls?

## 4.1 Microkernel Security Controls

This section discusses microkernel security controls present in the Mach, DTOS and LOCK6 systems.

### 4.1.1 Mach

Information in this section is taken from the paper, Providing Policy Control Over Object Operations in a Mach Based System [17] by Spencer E. Minear.

As discussed in Section 2.3.1.1, Mach uses its IPC facility as the focus of all system operations. An important question, then, is the extent of control provided through the IPC facilities.

Mach provides one primary control mechanism which is based on a *capability* concept. From the viewpoint of a task, a port is a task specific name called a *port right*. The task-specific port right embodies the capability (rights) that the task has to the port named by the port right. In Mach, however, the range of capabilities embodied in a port right is limited. Each Mach port right represents the capability to access one or more of the following kernel-supported IPC-related operations:[1]

- Send,
- Send-Once, and
- Receive

[1] A single port right can define either a send and/or a receive right, or a send-once right.

The following related operations deal with the transfer of port rights between tasks:

- The holder of a send right can, through the use of a send operation on any send or send-once right, have the kernel either move or duplicate the send right to the receiver of the message.
- The holder of a send-once right can, through the use of a send operation on any send or other send-once right, have the kernel move the send-once right to the receiver of the message.
- The holder of a receive right can, through the use of a send operation on any send or send-once right, have the kernel create a send or send-once right for the receiver of the message, or move the receive right to the receiver.

There are two problems, each discussed in more detail below, with these existing control mechanisms. The first is the limited control over the transfer of port rights and the second is the complete lack of control over the object-related services. These problems are not associated with capabilities in general. The required characteristics of capabilities for use in secure systems were outlined very clearly by Karger and Herbert [11]. Previous work has provided designs for capability machines that do deal with the limitations present in the Mach implementation. Examples are provided by HYDRA [29], SCAP [12] and ICAP [10]. A common element in other systems is that a capability is necessary, but not sufficient, to gain access to an object. The fundamental fault in Mach is that possession of a capability is sufficient for *full* access to *all* operations on the associated object. This makes it more difficult to build a secure system on Mach or on any other microkernel whose IPC lacks these control capabilities.

The intent of the DTOS work is to integrate the concepts present in these other capability machines into Mach. The system utilizes the improved Mach kernel as the base for a Unix operating system controlled by an underlying mandatory control policy.

The following two sections discuss in more detail the problems with the the existing Mach control mechanisms.

4.1.1.1 Port Right Transfer The primary problem associated with the rules for controlling the transfer of port rights is the complete lack of kernel-provided mechanisms or facilities to verify that once a transfer is complete, the operational state of the system is still in agreement with the system's security policy. This is particularly dangerous to both secure and safety-critical systems. It means that the kernel is unable to identify or stop a task from accessing a port via a port right it obtained as a result of an error or via malicious action. Applications are left to resolve this problem themselves without assistance from the kernel. One design technique that can be used is to inject a layer of indirection in the use of all IPC operations. For example, in a normal Mach environment two tasks that are allowed to communicate may do so directly with the use of IPC. This means, however, that the sending task can transfer any right it holds, either intentionally or by accident, to the receiving task. If an application needs to assure that rights cannot flow from the sender to the receiver, then it is necessary to have an intermediary task to filter messages to stop the transfer of port rights that violate the system's security policy. This approach can lead to sufficient control for many applications but may result in undesirable performance penalties and increased complexity in the application.

4.1.1.2 Service Control Because the Mach port right control facilities have no association with object services accessible via a port right, Mach provides no direct control over object services. If an application needs to provide control over individual object services, it must address the problem by binding groups of object services to ports, essentially subdividing an object. The application can then attempt to control access to the services by controlling the distribution of port rights to the various groups of services.

An example of the use of this approach can be seen in the design of the Mach kernel itself. One of the kernel objects is the kernel itself, referred to as the *host* object. The range of operations available for the manipulation of the host object, however, are split into two groups: the privileged operations, like **host_reboot** and **host_set_time**, and generally available operations, like **host_info** and **host_get_time**. The designers of the kernel recognized that it would be necessary to control access to the kernel's privileged operations independently from the general operations. Thus, the host operations were split into the two groups with the privileged operations bound to the *host-privilege* port and others to the *host port*.

There are two undesirable aspects of this approach for controlling services. The first is the lack of flexibility. A grouping that is correct for one application and security policy might be incorrect for another application or security policy. The lack of flexibility of the grouping approach is particularly evident in the grouping of task object services. In total, there are about 45 different task services available on a task port and there is no ability to control access to these services individually. Thus, a holder of a send right to a task port has implicit permission to all 45 task related services. It is an all or none situation.

The second undesirable aspect of this approach is that it does not scale well. If it were possible to assign the operations to different ports, the result might be a larger number of ports, especially in the case of objects with many services such as the kernel's task object. This leads to complexity of the control aspects of the design. Unnecessary complexity of any type in any system is undesirable. In the case of secure and safety-critical systems, unnecessary complexity is especially undesirable and must be avoided wherever possible.

In the following section, we describe the security additions that were made to Mach as part of the DTOS program.

### 4.1.2 DTOS

Information in this section is taken from the papers Providing Policy Control Over Object Operations in a Mach Based system [17] by Spencer E. Minear and Developing a "Policy Neutral" Control Policy for a Microkernel [6] by Todd Fine and Edward A. Schneider.

#### 4.1.2.1 Overview of DTOS Microkernel and Security Server Interaction

As mentioned in Section 2.3.1.2, a current focus of the DTOS program is to add support for a wide range of access control policies to the Mach microkernel. This is being accomplished by inserting control logic in the microkernel and adding a user space security server that performs security computations for the microkernel. In other words, the processing of each microkernel request is being modified to request a security computation by a security server before providing a service. This is illustrated in Figure 2.

When requesting a security computation, the kernel must provide information indicating the task that is requesting the service and the entity upon which the service is to operate. Other than regions in virtual address spaces, all entities in Mach are represented by ports. Thus, it suffices for the kernel to associate security information with each task, port, and memory region. When requesting an access computation, the kernel provides the security information

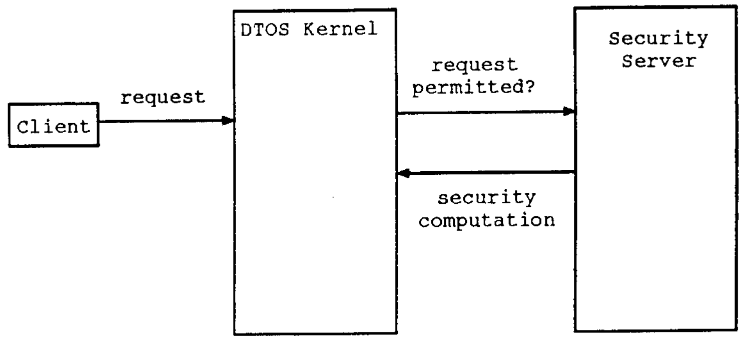

Figure 2: DTOS Kernel and Security Server Interaction

for the accessing task and the security information for the entity upon which the operation is to be performed. In response, the Security Server provides an access vector indicating which operations the accessing task may perform on the entity. Although the Security Server could simply respond with a yes/no answer as to whether the operation is permitted, the entire set of permitted operations is returned for efficiency. By caching the returned access vectors and consulting the cache before requesting computations from the Security Server, the kernel can avoid interactions with the Security Server when the necessary information is in the cache.

The security information that a security server needs to make access computations depends on the particular policy implemented by that security server. For example, a security server enforcing an MLS policy makes its decisions based on the security levels of the accessing task and the accessed entity. However, having the microkernel provide security levels to the Security Server would be incorrect since it would hard code into the microkernel that each entity has a security level. To be truly policy flexible, the microkernel cannot contain any policy specific information. Thus, the microkernel associates opaque labels called *security identifiers* (SIDs) with each kernel entity. Only the Security Server can interpret the meaning of a SID. The Security Server does so by recording the mapping between a SID and the *security context*, which defines the meaning of the SID. In the case of a MLS policy, a security context might consist of simply a security level. In the case of a Type Enforcement [2] policy, the security context associated with a task SID might contain only a domain, while the security context associated with each of the other SIDs might contain only a type.[2] The level of indirection provided by SIDs allows the same microkernel to be used regardless of how the Security Server interprets SIDs and makes access decisions.

The next section discusses in more detail how DTOS added security control to Mach.

#### 4.1.2.2 DTOS Security Control

As mentioned in Section 2.3.1.2, the prototype being developed on the DTOS program by Secure Computing Corporation consists of a modified Mach kernel and an external Security Server. The separation of policy decisions done in the Security Server from enforcement done in the kernel has proven successful in the LOCK system [21] and was discussed in the context of a Unix system by Walker, Kemmerer and Popek in [27]. The prototype attempts to resolve the limitations in the base Mach control mechanisms that were outlined in Section 4.1.1. To accomplish this, the prototype has added two new control mechanisms not available in the base Mach kernel and added a new interface

[2] Type Enforcement controls subject-to-subject access on a domain-to-domain basis and subject-to-object access on a domain-to-type basis. Thus, the security information needed to make decisions consists of domains and types.

to the kernel. The additions are, respectively:

**IPC Control** — The prototype provides expanded control over all aspects of port right manipulations. This allows the prototype's kernel to enforce policy-directed control over the transfer of port rights as well as over the use of the basic IPC operations.

**Object Service Control** — The prototype extends the port right capabilities to define policy directed control over the individual object services. The prototype kernel provides control over the individual services related to all kernel objects.

**Security Server** — The prototype implements a new interface between the Mach kernel and an external Security Server. This allows very strong separation between the enforcement mechanisms and the security-policy decisions. It allows the prototype system to ensure that all port right usage is in agreement with the current state of the security policy at the time of each usage. It also allows the system to localize the security policy in a single system element.

These additions address the control limitations discussed in Section 4.1.1. The two additional control mechanisms ensure that all system operations are subject to control. The new interface provides for the flow of control information from a mandatory security policy implemented in the Security Server to the enforcement mechanisms in the kernel and non-kernel object servers.

The general approach used in the prototype to add these new control mechanisms is based on the concept of a *security fault*. The security fault concept and its implementation within the prototype are very similar to that of Mach's page fault processing and the use of external pagers to implement memory objects. A security fault occurs when a task attempts to use a port right for which there is no readily available access-permission information. In response to the security fault, the kernel interacts with the Security Server to obtain the relevant permission information. To minimize the costly interactions between the kernel and the Security Server, the kernel caches the permission information, in the form of *access vectors*, for future reference, just as the kernel caches data to minimize interactions with pagers.

To implement the new control mechanisms following the ideas laid out by the security fault concept, five specific types of changes were made to the Mach kernel:

1. The addition of identification information on kernel objects to support the policy-based access decisions,
2. The addition of permission checks and security-fault detection in the kernel's IPC processing software,
3. The addition of permission checks and security-fault detection in the kernel's object service processing software,
4. The addition of an access vector cache to minimize interactions between the kernel and Security Server, and
5. The extension of the kernel interface:
   - The addition of a new interface for the Security Server. It allows the kernel to obtain object access-permission information from the Security Server and allows the Security Server to invalidate previously granted permissions.
   - The extension of the existing IPC facilities to provide identification and permission information to external object servers along with a service request. The identification and permission information are available to the kernel IPC services from the kernel's access vector cache.

Figure 3 shows the structure of the extended Mach kernel and its interaction with the Security Server. It shows that the permission checks are done in the IPC processing to control the use of all IPC related services. It also shows that permission checking is done in the kernel's service processing software to provide control over individual object services. Before a kernel object's server initiates a requested service, both of these permission checks must be passed successfully.

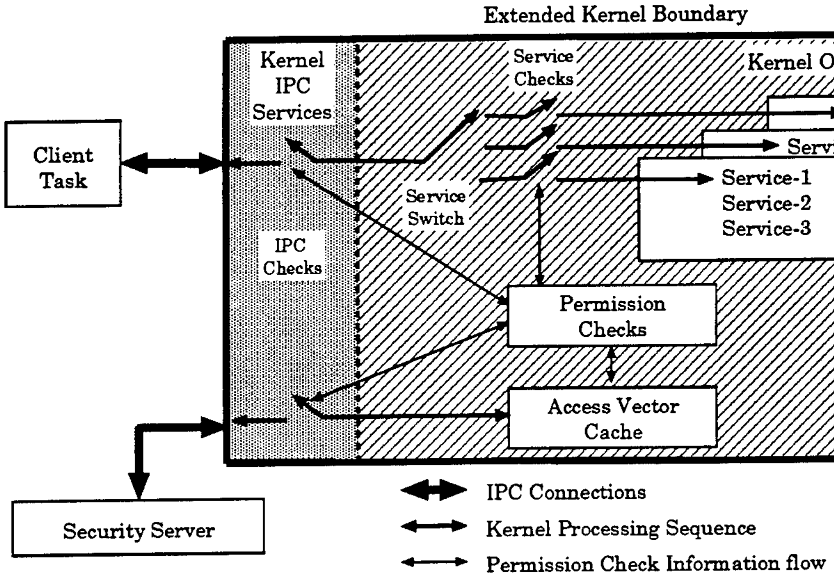

Figure 3: Kernel Control Mechanisms

Searches in the kernel's access vector cache are based on a pair of identifiers, bound to the relevant kernel objects. The first identifier is the *Source Security ID* (SSID) which embodies the control-relevant identity of the task making the request. The second identifier is the *Target Security ID* (TSID) which embodies the control identity of the object being accessed. When no entry is found in the cache, the current thread takes a security fault and the kernel makes a permission information request to the Security Server task. The kernel provides the (SSID, TSID) pair of identifiers and the permission being checked to the Security Server. The Security Server responds with the required access vector information that reflects the permissions based on the current state of the system's security policy.

Figure 4 shows the flow of identity and permission information from the kernel's access vector cache to a receiver of a request. The IPC processing binds the requester's SSID and access vector to the request message. Because the message receive operation is a direct communication between the kernel and a server, the object server can rely on the integrity of this identity and permission information and make object-specific policy-enforcement decisions as required. With the assumed proper operation of the kernel, the information is correct and was provided by the system's Security Server. Each object server, whether in or out of the kernel, has a very simple enforcement operation that is easy to test and verify. Other enforcement related processing in the kernel is straightforward processing which binds information to relevant structures and reports the bound information correctly.

The Security Server is the central point in the system where all policy decisions, the most complicated and critical part of any secure or safety-critical system, are made. If the system's

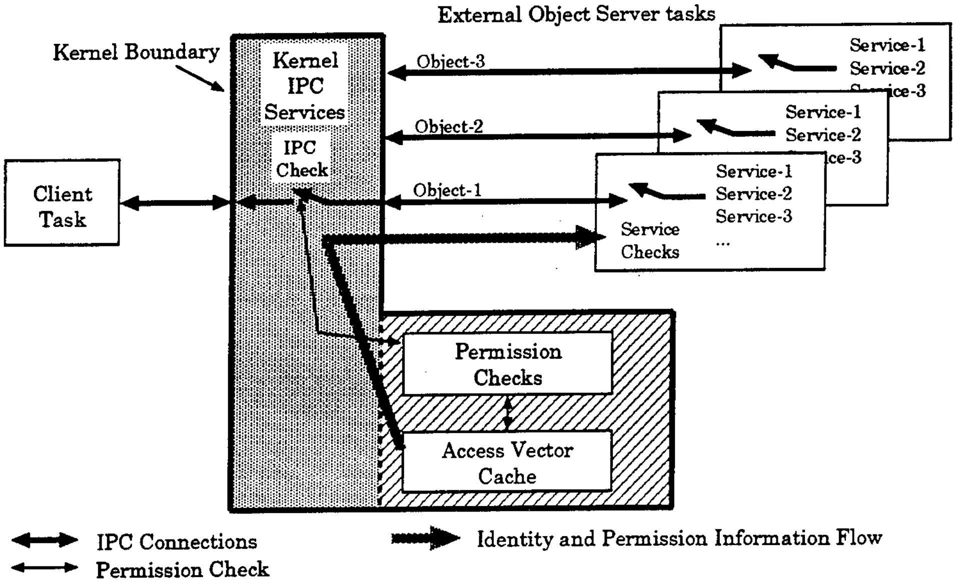

Figure 4: External Object Server Control

security policy cannot be assessed for correctness in the context of the single Security Server, it is highly unlikely that the security policy could be assured correct in any other implementation.[3]

The following sections discuss various aspects of the specific changes that were made to the Mach kernel.

4.1.2.3 Additional Identifiers To support the split of enforcement from policy decision, it is necessary to bind identifiers to all kernel objects. Within the Security Server, the identifiers are bound to policy-specific attributes such as user name, data type, security level, etc. In the kernel's policy-enforcement operations, the identifiers are simply numbers associated with objects that are to be passed as parameters to permission checks. This makes the split between enforcement and policy very clean. The kernel and other server enforcement software is completely independent of the security policy. This makes it possible to use the same kernel and applications in systems that must operate according to very different security policies.

The list of objects that were labeled with security identifiers includes:

- Tasks (SSID),
- Ports (TSID), and
- Memory Cache Objects (TSID).

4.1.2.4 Access Vectors For the kernel or any object server to actually enforce a policy decision, it is necessary for the enforcement software to have access to current permission

[3] We recognize that there are multiple aspects of many system control policies and that not all of them should be centralized in all systems. What we are referring to here is the basic security policy which defines the fundamental operation of a system. Specific servers are free to extend this base policy. For example, a file system server is the proper place for a Discretionary Access Control (DAC) policy such as Access Control Lists (ACLs).

information at each point where a security fault may occur. The permission information is provided in the form of an access vector which is computed based on the relevant (SSID, TSID) pair of identifiers. Each access vector defines the current state of permissions that the SSID has to all operations supported by the object bound to the TSID.

The structure of access vectors within the prototype is based on the two aspects of the control mechanisms: the IPC and object specific services. Figure 5 shows this basic two-part structure of an access vector. The fields in the IPC portion of the access vector are common to all access vectors because all services are accessed via IPC operations. The service part of the access vector, however, is viewed as a union of all possible object-specific access vectors. The addition of other service vectors has no impact on the kernel as service checks are always done in the context of the specific object.

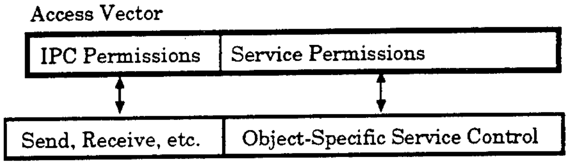

Figure 5: Access Vector Structure

This approach results in very simple, easy-to-assure enforcement software. It consists of a simple test of the appropriate field of an access vector. This approach is also very easy to extend to include the specification of control over application level objects as additions to the system's security policy.

Because all access vectors include the associated IPC permissions, the kernel continues to be the enforcer of the IPC permissions for all object accesses on the system. This means that the kernel—the unbypassable system element—is capable of enforcing the system security policy's definition of allowed task interactions.

4.1.2.5 Interface Extensions Another aspect of the prototype kernel work is the modification of the Mach kernel interface. A key requirement levied on the prototype work was that all changes to the Mach kernel interface would maintain backward compatibility with the existing interface. To satisfy this requirement, all changes to the interface are in the form of one of two types of extensions:

1. Extensions to make security relevant information visible to tasks, and
2. Extensions to support the kernel-Security Server interactions. These interactions resolve security faults and respond to policy state changes within the Security Server.

In making security relevant information visible to tasks, eight new entries were added to the kernel interface. Each is closely associated with an existing kernel interface and differs only in that extra parameters are accepted or provided. The additional entries are:

- Allow the creation of kernel entities, tasks, ports, and memory cache objects with specified identifiers, for example **task_create_secure** and **mach_port_allocate_secure**.
- Allow applications to obtain identifier and access information about kernel entities, for example **mach_msg_secure** and **mach_port_type_secure**.

All of the existing kernel interfaces remained syntactically the same, though their operational semantics may be affected by the policy denying the required permissions.

The extensions to support the kernel-Security Server interactions consist of one outcall from the kernel to the Security Server and additional kernel services used by the Security Server. The single new outcall is used by the kernel when it needs to obtain an access vector to complete the processing for a security fault. The thread causing the security fault is forced to wait until the response is provided by the Security Server. The kernel provides the Security Server with the appropriate (SSID,TSID) pair and indicates which permission is being checked. The Security Server responds with the same pair of identifiers and the current state of the associated access vector. In addition, the Security Server passes back a *cache control vector* and a *notification vector*. The *cache control vector* is used by the Security Server to instruct the kernel not to cache certain parts of the returned access vector. It is needed to support security policies in which accesses can be revoked. The *notification vector* is used by the Security Server to instruct the kernel to send a notification if a certain permission is being checked. It is needed to support the use of security policies which determine the current permissions based on the history of previous accesses to the associated objects. The response from the Security Server is sent on the thread's Remote Procedure Call (RPC) reply port which is controlled by the kernel.

Two additional services were added to the kernel's host object, accessible on the generally available host port.[4] These additional services allow the Security Server to:

1. Register the port which the kernel uses to send permission requests to the Security Server, and
2. Tell the kernel to flush all or part of its access vector cache.

The Security Server uses the first new service to notify the kernel that it is operational and to identify the port to use for sending permission check requests. Prior to the point in time, during system startup, when the Security Server becomes operational, the kernel must be able to make permission decisions on its own. As part of the prototype development, a list of the permissions for the operations done during system startup was developed and integrated into the kernel as the initial state of the access vector cache. This means that the initial operation of the system is done in agreement with this limited security policy statement. This part of the design is important to help establish the integrity of the system's initial state which is a key issue in the operation of any secure or safety-critical system. The system could disable all permission checks until the Security Server is operational. It is better, however, to specify correct operation even during startup and ensure that permission checking is always enabled.

The Security Server uses the second new kernel service to control the state of the kernel access vector cache. This facility allows the prototype kernel to support Security Servers which implement a variety of dynamic security policies where access permissions change during the operation of the system for any number of policy-controlled reasons.

### 4.1.3 LOCK6

This section discusses the LOCK6 microkernel security controls. Many aspects of the LOCK6 system's control approach were motivated by research done during the DTMach and DTOS

[4] With the base Mach control concept these operations would have to be split between the host privilege port and the host port. The prototype relies instead on the policy-defined control to specify which tasks in the system are allowed to request the specific operations.

programs described in Section 2.3.1.2 and Section 4.1.2. The general control approach is very similar to that described for the DTOS system. The implementation details are quite different, but the general control philosophy for both can be summed up in the following general rules which were established in the original LOCK work:

- Build the system on the smallest possible number of primitives implemented in the smallest amount of code. This makes it possible to carry out reasonable assurance on the real system.

- Integrate the control over the primitives at the same layer of the system where the primitive is implemented. This makes it impossible to bypass the control mechanisms.

- Provide strong separation between the policy decision and the enforcement of the policy decisions. This makes the system inherently more flexible. The implementation ensures that the operations it provides are controllable and that the security policy can focus on defining the minimum set of permissions required for the system to operate.

In LOCK6, as in DTOS, since nearly all system operations are implemented on the IPC facilities, this means that the focus of the system control is on providing control over the IPC facilities.[5] The issue of what to control is determined by the operational semantics of the LOCK6 IPC facilities and the view that if there is a kernel operation that is accessed by any user code, then the system's control mechanisms must allow the policy writer to specify which subjects are allowed to access the service and which are not. To make this possible, each process is given a Subject Security Context, (SSC), and each IPC channel is given an Object Security Context, (OSC). Within the LOCK6 kernel, the policy enforcement processing makes a permission check on each object access to determine if a given operation is to be permitted. The enforcement logic provides the appropriate SSC and OSC values to the security policy which returns the relevant permissions information.[6] If the permission is granted, the requested operation's processing continues. If the permission is not granted, the requested operation's processing is terminated at that point.

As in many message based systems, the LOCK6 IPC channels may be used by applications to represent specific abstract objects which have object specific services implemented by an object server process. The LOCK6 control approach utilizes the extensible control concept defined in the DTOS program. This allows a system security policy to be extended to include the specification of control over the object specific services when the object service is added to the system. The object specific policy enforcement logic is provided in server interface library software which is generated from a server interface specification. When the IPC channel is established, the client subject's permissions to the IPC channel and associated object are bound to the channel. On each use of the channel, the LOCK6 kernel enforces the IPC services and binds the permission information to the request. The object server interface library software enforces the object specific aspect of the permission based on the permission information provided by the kernel. This provides a very efficient means of providing permission information directly to the server.

[5] The operations that are not accessed via IPC are limited to direct system traps which are controlled utilizing the basic control services as in the IPC operations.

[6] For efficiency of operation, the permissions are cached so that it is not necessary to access the actual security policy on each check.

## 4.2 Microkernel Security Controls Applied to DBMS Systems

This section discusses how DTOS microkernel security controls might be applied to DBMS systems. On this project, two types of DBMS access control were investigated: IBAC and MAC. The objective was to understand how the DTOS access control mechanisms can be used to develop a high assurance version of DBMS access control.

The fundamental requirement for using the DTOS access control mechanism to control access to an object is that the object must be registered with the DTOS kernel. The mechanism for registering an object is the mach_port_allocate request. When a port is allocated to an object, the kernel is responsible for creating the port and controlling access to it; the security server is responsible for making security decisions based on the security context of the requesting task and the target object; and the application server is responsible for enforcement of application specific security decisions made by the security server. This assignment of responsibilities is illustrated in Figure 6. The application server is also responsible for supporting any additional application specific security checks that the security server is not aware of.

For the DBMS example, the application server is that portion of the DBMS that manages DBMS objects. In the following sections we explore what these objects and their application specific security attributes might be. For this approach to work, it must be the case that all requests to access the objects are made with the security context of the client. How this might be done is discussed in a later section.

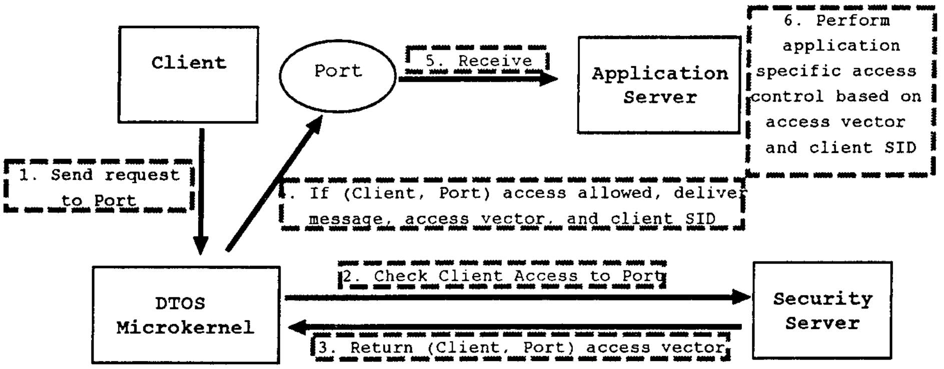
Client
Port
5. Receive
Application Server
6. Perform application specific access control based on access vector and client SID
1. Send request to Port
. If (Client, Port) access allowed, deliver message, access vector, and client SID
2. Check Client Access to Port
DTOS Microkernel
Security Server
3. Return (Client, Port) access vector

Figure 6: DTOS Security Mechanisms.

The Security Server decides what accesses a client has to a specific port. The access vector returned contains an IPC portion, that the microkernel uses to determine if the message can be sent, and an application specific portion, that is configurable for each application and that the application uses to enforce its own policy.

### 4.2.1 Relational DBMSs

This section discusses how DTOS security controls can be applied to relational DBMS systems.

#### 4.2.1.1 IBAC for Relational DBMSs

The standard IBAC controls provided by a commercial RDBMS are described in Section 2.3.2.

The DTOS access control mechanisms could be used to implement a higher assurance IBAC policy that controls access to the DBMS named objects mentioned in Section 2.3.2: tables, views, stored procedures, sequences and packages. Whenever any of these objects is created, the DBMS application server would be responsible for requesting that a corresponding port be allocated for the object, and for maintaining the binding between a named object and its associated port.

When a client needed to access one of these named objects, the access would be via the assigned port. There are two levels of granularity at which the client's access to the DBMS named objects can be controlled.

At the kernel level, domain and type attributes can be used to enforce course grained control. Named objects with certain types may not be accessible at all to clients in certain domains. That is, no send/receive rights to the port and hence no access to the object would be allowed. For example, accounting types could be set up that accounting domains would have access to, but DBMS clients in other domains would not. This feature would support certain types of roles that require portions of the database to only be accessible from those roles.

To provide this feature in an effective manner, the microkernel OS would have to support adding domains and types dynamically, and for each domain and type added, updating the security policy in the Security Manager appropriately. The current version of DTOS does not provide this capability. In addition, the client must be running in a domain that carries its security context, that is, a generic server domain will not work. Since multithreaded servers are now commonly used, a mechanism to dynamically change the security context of a process would be most appropriate. As the server changed threads, its security context could be changed to reflect the security context of the client associated with the new thread. DTOS does not currently support this capability.

Any more fine-grained IBAC control cannot be performed by the DTOS kernel control mechanisms. The DBMS application server must provide this control. At the level of the DBMS, the operations that the application server controls would be:

- for tables: ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT, UPDATE
- for views: DELETE, INSERT, SELECT, UPDATE
- for stored procedures: EXECUTE
- for sequences: ALTER, SELECT

The application specific bits in the DTOS access control vector would be defined based on these operations as shown in Figure 7.

DBMS DAC system privileges would have to be treated in a slightly different manner since these privileges are associated with a database rather than a named object in the database. One approach might be to define the database itself as an object and register it with the Security Server. When a user attempted to perform a system privileged operation, the application server would check with the Security Server asking for the user's permissions to the database object. The application specific bits of the access vector returned would then contain the user's system privileges to the database.

Since the IBAC policy can be changed at the discretion of the owner of an object or another user who has been granted the appropriate privilege, a mechanism must be included that

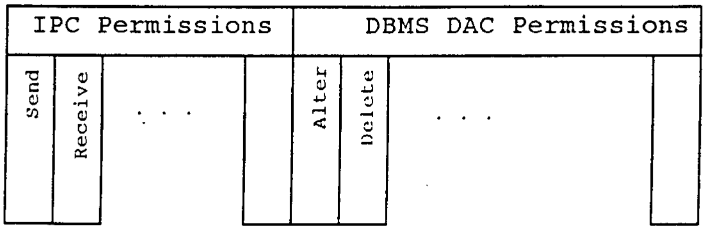

Figure 7: DTOS Access Vector for IBAC Support
The application specific bits of the access vector contain the IBAC permissions that the client has to the particular object.

allows the Security Server to be notified of modifications to the policy. For example, if the DBA adds a user and decides to allow that user access to certain named objects, then the Security Server must be notified so that it can update its tables to allow these accesses. This is true for both users and roles. (The policy would be that a user has access to an object if the user is specifically identified as having access or is in a role that has access.) At this point, conceptually the Security Server must maintain a matrix of allowed accesses that is indexed by subject security id (including domain, user, role, ...) and object security id (including name, type, ...). If IBAC is to be supported by the Security Server at this level of granularity, then there is potentially one column for every user on the system and one row for every object on the system. This could result in an extremely large and sparsely populated matrix. Techniques would have to be introduced into the Security Server to handle such data structures efficiently.

Rather than implement the entire IBAC policy using the Security Server, it might be possible to identify portions of it that are most critical, and change the least, such as DBA permissions, and just use the Security Server for these.

What has been gained by putting this IBAC control matrix into the Security Server rather than keeping it in the DBMS server? The main benefit is that the DBMS Server is simplified and the access control decision software is isolated so that it can be more easily assured. The DBMS Server must still enforce the access control decision, however, and a subverted DBMS Server could completely ignore the decision supplied by the Security Server. To prevent this from happening, the DBMS Server must be assured to always enforce the decisions correctly. If the DBMS server is a large, complicated piece of code, this may be difficult to do. One approach might be to decompose the DBMS Server into a component that accesses the data and a component that supplies the other DBMS functionality. The access component could be smaller and could be assured to properly enforce any access decisions. The access component would have to include things like locking capability, versioning, and concurrency control. There would also have to be a mechanism that passed the access component the correct security context of the user client making the request. This could not be passed via an untrusted portion of the DBMS. The point to note is that even if you break the DBMS Server up into smaller components, you must still assure any components that have access to data that is released to a client or received from the client to be entered into the database. These components could conceivably store data and pass it out to another client that did not have access. For example, the Oracle DBMS managers might be a start at providing some decomposition, but a communication mechanism that did not use System Global memory would have to be developed.

4.2.1.2 MAC for Relational DBMSs In current COTS Secure DBMS products, security levels are assigned on a row-by-row basis. That is, each row in a table has an associated level. The table itself becomes a multilevel object. The DTOS control mechanism does not directly lend itself to supporting a policy like this since the objects that are being controlled in this case are rows and assigning a port to each row in the database is not feasible or practical.

Partitioning tables by row level in the manner that is used by Oracle's OS MAC mode may be a possibility. However, since the DTOS kernel only enforces controls on send/receive access to a port, the kernel cannot provide any help in enforcing a MAC policy on the DBMS objects. The difficulty is, as mentioned before, the DTOS kernel has no knowledge of what can be sent or received via the port. If you have send access on a port that represents an object, then the kernel does not know if you are using that send to read the object or modify the object. It is up to the DBMS application server that maps objects to ports to enforce the semantics of the application. Partitioning the tables according to level might help the DBMS server enforce the policy, by, for example, making use of the Security Manager to make access control decisions, but it is still up to the server to enforce the decisions made by the Security Manager.

A form of non-MLS mandatory access control could be supported by DTOS using types and domains to enforce strong separation between databases. Each database would have its own separate types and domains. Only users in the appropriate domains would be allowed to access the corresponding database objects. In addition, roles that involve read-only access to the database could be enforced in this manner. This type of separation has been successfully demonstrated on the LOCK DBMS program [22]. However, this would require additional support in DTOS for dynamically creating new domains and types. In particular, it would be useful to be able to create domain/type templates for a subsystem that could then be instantiated whenever a new subsystem was created.

### 4.2.2 Object Oriented DBMSs

The issues involved in supporting MAC for an MLS OODBMS, such as TOP, are very similar to the issues involved in supporting MAC for a RDBMS. Just as rows can have separate levels and tables can thus be multilevel, attributes in an object can have separate levels and objects can become multilevel. In TOP, the object can be partitioned into its single level L-instantiations which are then stored at their own level. For the DTOS control mechanisms to be able to support such an architecture, the L-instantiations of each object would have to be treated as full objects in DTOS and given their own security ID. The Security Server would then be able to make access decisions based on the client user's security context and the level of the various L-instantiations.

As shown in Figure 6, the Object Manager would still need to enforce these decisions. In TOP a user is presented with views of the data in the database that are appropriate for the user's level. The advantage of an OODBMS is that the code that constructs these views could run in the context of the user. This would allow the user's security context to be attached to all requests to view an object so that the Security Server would have the right context for making decisions. Such an architecture might provide a much easier and more natural way to develop a high assurance MLS DBMS that the relational approach.

DAC for OODBMSs could be supported in a manner similar to that discussed for RDBMSs. Again the advantage in the OODBMS approach is that the code that requests access to objects will most likely already be running with the user's security context so that requests for access can be properly addressed by the Security Server. The security server would need to be aware of each type of object and the methods that can be invoked on that object. Flexibility would need to be added to the Security Server to support adding new objects with new methods.

*Section* 5

# Lessons Learned and Future Directions

This section will discuss lessons learned on the SDDM program and possible directions for future research in areas related to microkernels and DBMSs.

## 5.1 Lessons Learned Summary

The following points summarize the main conclusions relative to using DTOS control mechanisms for a DBMS.

- The DTOS control mechanisms work at the level of granularity of objects. In particular, this implies that to be able to use the DTOS control mechanisms to control access to a data element, the data element must be contained in an object that has the same security context as the data element. Objects in DTOS are defined by allocating a port to represent the object. If the security context is relatively fine-grained, this may require many small objects and corresponding ports.
- Object granularity is determined by the policy that you want to enforce.

  For the DTOS control mechanisms to apply to a relational DBMS IBAC policy, the objects might be tables, views, stored procedures, sequences and packages. That is, each table is an object and so on. For the DTOS control mechanisms to apply to a DBMS MAC policy, an object could only contain data elements that are all at the same level. Note that if we support both policies, then table, sequence and package objects might have to be partitioned into smaller objects that are single level. Views and stored procedures could receive a level and would not have to be split. To maintain global table constraints, it would be necessary to have some trusted subject that has access to all items in the table. Splitting a table and then having to put it together again to check global table constraints could be a major performance hit.
- The DTOS kernel can only control access to an object based on rights to the port associated with the object. These port rights only control whether the requester can use the port to send/receive messages to/from the object. The particular service requests that can be sent are object specific and control on whether the requester can make a particular service request is enforced by the Object Manager (that is, the application server) for the particular object.
- The DTOS control mechanisms work at the level of granularity of processes. In particular, access to an object by a DBMS process running in its own security context, but in response to a request from a DBMS client, cannot be controlled by DTOS based on the security context of the client.
- The object manager needs to enforce the bulk of the access control policy. If the system is to have high assurance, this implies that the object manager must be small or else decomposable into a portion that enforces the policy and can be highly assured and other less-assured components.

To allow an application server to make full use of the DTOS control mechanisms and the services provided by the Security Server, a number of enhancements would have to be made to the Security server. These include:

- To support OODBMSs the interface to the Security Server would need to be enhanced to allow the application to define new types of objects with new methods associated with them and then to define the security policy that should be enforced on these objects.
- To support a policy like a DBMS DAC policy via the Security Server would require support in the Security Server for storing and manipulating sparse access control matrices.
- A less ambitious enhancement would to allow templates for Type Enforced subsystems to be defined that would allow the application to create separate instances of the subsystem that were protected from one another via Type Enforcement. This might be useful, for example, for strong separation of databases.

## 5.2 Future Directions

This section will discuss future directions of the technology discussed in this report.

### 5.2.1 DBMS Development

Development of a research prototype high assurance MLS DBMS on the DTOS microkernel is currently underway at Penn State University under the direction of Dr. Thomas Keefe [28]. This work includes architecting a DBMS system for the MLS environment and development of several trusted servers, including a trusted buffer manager, a trusted scheduler and a trusted log manager. Control of covert channels is a major concern on this effort, and it should be noted that the DTOS control mechanisms do not provide any help in this area.

This research should provide additional insights into the adequacy of DTOS, and its predecessor Flux/Fluke, controls for DBMS systems in the future.

### 5.2.2 Web-based DBMS Servers

A future trend in the development of distributed DBMS systems is the use of web-based technology to present the interface to the DBMS system. Research into the possibility of adding controls to the web-based server to increase overall database security should be conducted.

### 5.2.3 Java

The Java Platform [13] is being developed by JavaSoft, a division of Sun Microsystems. Sun is publishing the specifications for the components of the Java Platform to encourage its adoption as an open solution for object-oriented, distributed networking.

Java is being widely adopted for Internet/Intranet applications.

The Java Platform has three components:

- the Java Language
- the Java Virtual Machine
- the Java Application Programming Interface (API)

Programs written in the Java language are compiled to an intermediate bytecode that the Java Virtual Machine understands. The Java Platform supports a number of built-in security features:

- Java is a "safer language" than C or C++.
  - Java is a strongly typed language. This allows many potential errors to be identified at compile time.
  - Pointer arithmetic is not allowed. In particular, it is not possible to access arbitrary memory locations by redefining a pointer, either accidentally or on purpose.
  - Garbage collection is performed automatically. This prevents "memory leaks" from corrupting the system.
  - Array bounds checking is done at runtime. Hence, it is not possible to overflow a stack and end up somewhere in memory where you don't belong.
- The Java runtime environment includes a verifier that can be run over Java bytecode to ensure that the bytecode is "correct". Correct in this context includes checking that the bytecode has the proper format and that certain language rules, such as typing rules, are not violated. The purpose of the verifier is to protect against malicious bytecode that has been obtained from an untrusted source.
- The Java Platform includes a Security Manager class that can be used to control the access to system resources such as files, network connections, threads, etc. Each Java application can define its own Security Manager that implements its own Security Policy. Java applets are constrained by the Security Manager of the application in which they are invoked, usually the Web browser.
- The Java API includes interfaces for encryption, digital signatures, and authentication.

Sun Microsystems is currently developing chips that implement the Java Virtual Machine in hardware. Sun has also defined a Java OS that consists of the Virtual Machine and certain base Java language classes. The Java OS runs directly on hardware and could be compared to a microkernel.

Since Java provides a complete object-oriented environment that can include its own base OS and since it has been promoted as a "secure" system, there are a number of research areas that would be of interest to investigate. These include:

- A comparison between the DTOS control mechanisms and the Java control mechanisms. In DTOS the Security Manager exists as a separate process. It provides the basic checks for the kernel and for any applications whose policies have been incorporated into the Security Manager. In Java the Security Manager is a class that presents an API that is called when security checks need to be made. There are certain security related system calls that always call the Security Manager, but since each Java application can define their own Security Manager, the results of these checks can be application dependent. In addition, each application can extend the Security Manager to provide whatever additional checks and APIs are necessary. This comparison would identify the strengths and weaknesses of each approach.

- An investigation of the "security" of the Java Platform as a "wrapper". Java code is executed within the context of the Java Virtual Machine. If the Virtual Machine is running on another OS and the Java code can either subvert the Virtual Machine or "breakout" of it, then malicious Java code might be able to subvert the security mechanism. This investigation would involve a complete security analysis of the Java Platform to determine its security vulnerabilities.

- A study of whether a security policy similar to a Type Enforcement policy could be implemented using the Java Security Manager. This might include defining classes in Java that have associated domains and types.

- A design of a Java Object Oriented DBMS that made use of the Java security features.

*Section* 6

# Notes

## 6.1 Acronyms

**ACL** Access Control List

**API** Application Programmer Interface

**DAC** Discretionary Access Control

**DBA** Database Administrator

**DBMS** Database Management System

**DTMach** Distributed Trusted Mach

**DTOS** Distributed Trusted Operating System

**IBAC** Identity Based Access Control

**IPC** Inter-Process Communication

**LOCK** Logical Coprocessing Kernel

**MAC** Mandatory Access Control

**MLS** MultiLevel Secure

**MMU** Memory Management Unit

**OODBMS** Object Oriented Database Management System

**OS** Operating System

**OSC** Object Security Context

**OSF** Open Software Foundation

**RDBMS** Relational Database Management System

**RPC** Remote Procedure Call

**SCC** Secure Computing Corporation

**SDDM** Supporting a Secure DBMS on the DTOS Microkernel

**SID** Security Identifier

**SNS** Secure Network Server

**SSC** Subject Security Context

**SSID** Source Security Identifier

**SSO** System Security Officer

**TCB** Trusted Computing Base

**TCSEC** Trusted Computer Security Evaluation Criteria

**TO7** Trusted Oracle 7

**TOP** Trusted ONTOS Prototype

**TSID** Target Security Identifier

**VMM** Virtual Machine Monitor

*Appendix* *A*

# Bibliography

[1] Martín Abadi and Leslie Lamport. Composing Specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[2] W. E. Boebert and R. Y. Kain. A Practical Alternative to Hierarchical Integrity Policies. In *Proceedings 8th National Computer Security Conference*, pages 18–27, Gaithersburg, MD, October 1985.

[3] E. F. Codd. A Relational Model for Large Shared Databanks. *Communications of the ACM*, 13:377–387, 1970.

[4] Oracle Corporation. *"Trusted ORACLE Administrator's Guide, Version 1.0"*. Number 6610-10-0392. Oracle Corporation, Redwood City, CA, 1992.

[5] Todd Fine and Spencer E. Minear. Assuring Distributed Trusted Mach. In *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, pages 206–218, Oakland, CA, May 1993.

[6] Todd Fine and Edward A. Schneider. Developing a "Policy Neutral" Control Policy for a Microkernel. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, November 1994. Submitted for publication.

[7] Bryan Ford and Mike Hibler. Fluke: Flexible $\mu$-kernel Environment Application Programming Interface Reference. Technical report, Department of Computer Science, University of Utah, Salt Lake City, UT, May 1996. DRAFT.

[8] Bryan Ford, Mike Hibler, Jay Lepreau, Patrick Tullmann, Godmar Back, Shantanu Goel, and Steven Clawson. Microkernels Meet Recursive Virtual Machines. Technical report, Department of Computer Science, University of Utah, Salt Lake City, UT, May 1996. DRAFT - UUCS-96-004.

[9] Bryan Ford, Mike Hibler, and Flux Project Members. Fluke: Flexible $\mu$-kernel Environment Design Principles and Rationale. Technical report, Department of Computer Science, University of Utah, Salt Lake City, UT, April 1996. DRAFT - CONFIDENTIAL - DO NOT DISTRIBUTE.

[10] Li Gong. A Secure Identity-Based Capability System. In *IEEE Symposium on Computer Security and Privacy*, pages 56–63. IEEE, 1989.

[11] P.A. Karger and A.J. Herbert. An Augmented Capability Architecture to Support Lattice Security and Traceability of Access. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 2–12, April 1984.

[12] Paul Ashley Karger. Improving Security and Performance for Capability Systems. Technical Report 149, University of Cambridge, Cambridge England, October 1988.

[13] Douglas Kramer. The Java Platform. Technical report, JavaSoft, Mountain View, CA, May 1996. whitepaper.

[14] Keith Loepere. *Mach 3 Kernel Interfaces*. Open Software Foundation and Carnegie Mellon University, November 1992.

[15] Keith Loepere. *OSF Mach Kernel Principles*. Open Software Foundation and Carnegie Mellon University, final draft edition, May 1993.

[16] Mary E. S. Loomis. *Object Databases:The Essentials*. Addison-Wesley, Reading, MA, 1995.

[17] Spencer E. Minear. Providing Policy Control Over Object Operations in a Mach Based System. In *Proceedings of the Fifth USENIX UNIX Security Symposium*, pages 141–156, Salt Lake City, Utah, June 1995.

[18] MITRE. MLS Microkernel DBMS Design Analysis. Technical report, MITRE, February 1996. DRAFT.

[19] NCSC. Trusted Computer Systems Evaluation Criteria. Standard, DOD 5200.28-STD, US National Computer Security Center, Fort George G. Meade, Maryland 20755-6000, December 1985.

[20] NSA. "Trusted Database Management System Interpretation of the Trusted Computer Systems Evaluation Criteria (TDI)". Technical Report NCSSC-TG-021, NCSC, Fort George G. Meade, MD, April 1991.

[21] O. Saydjari, J. Beckman, and J. Leaman. LOCK Trek: Navigating Uncharted Space. In *IEEE Symposium on Computer Security and Privacy*, pages 167–175. IEEE, 1989.

[22] SCC. LOCK DBMS Final Report. Technical report, SCC, 1996.

[23] Marvin Schaefer, Valerie Lyons, Paul Martel, and Antoun Kanawati. TOP: A Practical Trusted ODBMS. Technical report, ONTOS, Lowell, MA, 1995. whitepaper.

[24] Secure Computing Corporation. DTOS Kernel Interfaces Document. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, April 1995. DTOS CDRL A003.

[25] Michael Stonebraker. Operating System Support for Database Management. *Communications of the ACM*, 24(7), July 1981.

[26] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Inc., Englewood Cliffs, NJ, 1992.

[27] Bruce J. Walker, Richard A. Kemmerer, and Gerald J. Popek. Specification and Verification of the UCLA Unix Security Kernel. *Communications of the ACM*, 23(2):118–131, February 1980.

[28] Andrew Warner, Qiang Li, Thomas Keefe, and Shankar Pal. The Impact of Multilevel Security on Database Buffer Management. *To appear.*, 1996.

[29] William A. Wulf, Ellis Cohen, William Corwin, Anita K. Jones, Roy Levin, C. Pierson, and F. Pollack. HYDRA: The Kernel of a Multiprocessor Operating System. *Communications of the ACM*, 17(6):337–345, June 1974.

☆U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-47160

# *MISSION*
# *OF*
# *ROME LABORATORY*

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.